Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 32 (2014) 808 – 815

Procedia Computer Science

The 3rd International Workshop on Agent-based Mobility, Traffic and Transportation Models, Methodologies and Applications (ABMTRANS)

Agent-based simulation testbed for on-demand mobility services

Michal Čertický, Michal Jakob, Radek Píbil, Zbyněk Moler

Agent Technology Center
Faculty of Electrical Engineering, Dept. of Computer Science and Engineering
Czech Technical University in Prague, Czech Republic

Abstract

New markets for personalized and efficient transport are creating a need for on-demand mobility services. To rigorously analyse new control mechanisms for these services, we introduce an open-source agent-based simulation testbed that allows users to evaluate the performance of multi-agent, on-demand transport schemes. In particular, our testbed provides a framework to compare both centralized and decentralized, static and dynamic passenger allocation and vehicle routing mechanisms under various conditions; including varying vehicle fleets, road network topologies and passenger demands. Our testbed supports all stages of the experimental process; from the implementation of control mechanisms and the definition of experiment scenarios, through to simulation execution, analysis, and interpretation of results. Ultimately, the testbed accelerates the development of control mechanisms for emerging on-demand mobility services and facilitates their comparison using well-defined benchmarks.

Keywords: simulation; mobility; transport; multi-agent simulation; testbed;

1. Introduction

On-demand mobility services have the potential to bring significant improvements to personalized transport via efficient utilization of transport vehicles. In on-demand mobility services, vehicle routes and schedules are not fixed a priori; instead, they are dynamically adapted to best serve continuously incoming transport requests. Traditionally, on-demand mobility approach was used mainly for providing small-scale specialized transport services that required prior-day booking. In the past years, enabled by the widespread adoption of smartphones and ubiquitous internet connectivity, on-demand mobility approach has been increasingly utilised for general-purpose real-time ridesharing, taxi, or bus-on-demand services. Looking to the future, new autonomous, driverless vehicles, are set to lead to further growth in both scale and scope of on-demand mobility services.

* Michal Čertický. Tel.: +420-224-357-337; Fax: +420-22492-3677.
E-mail address: certicky@agents.fel.cvut.cz

doi:10.1016/j.procs.2014.05.495

On-demand mobility services are inherently multi-agent. This is due to a large number of geographically distributed passengers, vehicles and service providers which, while having their individual interests, have to coordinate and agree on how the passenger demand is served by available transport resources. As such, agent-based techniques are beginning to play an important role in passenger allocation and vehicle coordination[5].

The performance of on-demand mobility services with multi-agent based allocation and coordination crucially depends on two key factors: (1) the transport *control mechanism* used to allocate vehicles to passengers and to determine vehicle routes and timings; and (2) parameters of the *deployment scenario*, in particular the topology of the underlying road network and spatio-temporal structure of passenger demand. Understanding how these factors affect the performance of the transport system is essential for the development and deployment of on-demand mobility services. Due to the complex nature of demand-responsive transport systems, however, gaining such understanding is difficult.

Simulation modelling is an established approach for analysing the behaviour of complex socio-technical systems and is therefore also applicable for analysing demand-responsive transport systems. Unfortunately, out of the number of transport simulation tools, none is specifically tailored and consequently particularly suitable for simulation modelling of on-demand mobility services. In fact, within the broad family of *pickup and delivery problems*, in which on-demand services are a special case, benchmarking suites only exist for static freight transport vehicle routing problems[1]. To the best of our knowledge, no benchmarking tools exist for dynamic, passenger-oriented variants of the pickup and delivery problem. The key reason for the lack of benchmarking tools is that a simulation engine is required to rigorously account for temporal dependencies. This means that benchmarking on-demand mobility services is a significantly more challenging problem and cannot be solved by simple evaluation tools that are sufficient for static variants of the problem.

In this paper, we detail our new testbed – a rigorous and flexible benchmarking suite for on-demand mobility services. The testbed is based on our previous research in fully agent-based simulation modelling of transport systems[7]. It is built on top of a versatile transport simulation framework AgentPolis[8].

Our testbed is designed for two main purposes. The first purpose is performance assessment of on-demand mobility services prior to their deployment in new locations or under different conditions. The second purpose is testing and evaluation of novel control mechanisms, algorithms and on-demand mobility schemes. As such, our testbed can dramatically speed up the development and deployment of on-demand mobility services.

2. Related Work

The general idea of employing simulation testbeds to accelerate the development of multi-agent control mechanisms has already been put forward in the past.[13,9] Focusing specifically on transport, M. Horn employed an agent-based simulation, developed completely from scratch, to study operational characteristics of a multimodal transport system integrating conventional timetabled services and flexible on-demand mobility services.[6] Demand-responsive transport systems, and the impact of zoning vs. non-zoning strategies on them, were studied by Quadrifoglio and Dessouky.[14] Real-time taxi sharing schemes have been also evaluated using simulations. For example, D'Orey et al. used simulations to explore the trade-offs between the satisfaction of drivers and passengers,[3] while Lioris et al. aimed to provide an autonomous taxi-sharing service taking advantage of additional information about traffic conditions.[12] Cooperative on-demand paratransit services with emphasis on resource allocation were studied by Fu[4] and Jlassi.[10]

All the transport system simulations used in the works above were developed from scratch using conventional programming languages like Java or C++, since existing general purpose (such as AnyLogic[2]) as well as transport-specific simulation tools (such as MATSIM or SUMO[3]) have proven insufficient for simulation-based assessment of a wider variety of demand-responsive transport systems. Still, none of these simulations were able to deal with both static and dynamic transport demand at the same time, and to support both centralized and distributed control mechanisms, while allowing users to implement custom behaviour of passengers, drivers and dispatchers, based on their mutual communication. Also, they do not provide a benchmarking mechanism that would ensure identical conditions for different tested control mechanisms and an easy to use programming interface.

[1] http://www.or.deis.unibo.it/research_pages/ORinstances/VRPLIB/VRPLIB.html
[2] http://www.anylogic.com
[3] http://www.matsim.org, http://sumo-sim.org/

3. Testbed Overview

The proposed simulation testbed is built upon the versatile transport simulation framework AgentPolis, which provides abstractions, code libraries and software tools for building and experimenting with fully agent-based models of interaction-rich transport systems.

3.1. Fully Agent-Based Simulation Approach

In fully agent-based simulations, individual entities of a transport system are represented as autonomous agents with continuous, asynchronous control modules and the ability to interact freely with their surrounding environment and other agents. Such an approach reduces coupling and allows modeling scenarios in which agents adjust their plans at any time during the day based on their observations of the environment and/or communication with other agents.

The AgentPolis framework, which implements the fully agent-based approach, provides several dozens of modelling elements that can be used to build a wide range of simulation models. The modelling elements provided by AgentPolis are organized in a modelling ontology and can be grouped to three high-level categories:

- *Agent modelling elements:* The concept of the *agent* in AgentPolis is defined rather loosely in order to support modelling a wide variety of agents (e.g. `DriverAgent`). The behavior of agents is defined in terms of *activities* – reactive control structures implementing the logic determining which actions or nested activities the agent executes at a certain point in time or in response to sensor information or messages received from other agents (e.g. `DriveVehicle` activity). As part of their behaviour, agents may need to make decisions that require executing complex algorithms, including the ones that comprise the control mechanisms we want to evaluate. In order to promote reusability, such algorithms are encapsulated into so-called *reasoning modules*. In practice, the reasoning modules are Java classes (e.g. `DriverLogic`) that can be easily rewritten to implement a wide variety of algorithms, or even call external tools or solvers.
- *Environment modelling elements:* The environment models the physical context in which the agents are situated and act. It is represented by a collection of *environment objects*, each representing a fragment of the modelled physical reality (e.g. `Vehicle`), and *queries* that allow agents to be informed about the state of the environment and about the events happening during simulation execution (e.g. `PositionQuery`).
- *Interaction modelling elements:* Modelling complex interactions among the agents or between the agents and the environment is crucial for the analysis of dynamic transport systems. In AgentPolis, agent-environment interactions are realized by *sensors*, which process the percepts from the environment and atomic *actions* that provide a low-level abstraction for modelling how agents actually manipulate the environment (e.g. `MoveVehicle`). Inter-agent interactions are realized by a collection of *communication protocols*. Currently, the testbed provides `1-to-1 messaging`, `1-to-many messaging` and `auction` protocols.

3.2. Testbed Architecture

Although all the power and flexibility of the AgentPolis framework is accessible to the users of the testbed, it is hidden and only the relevant parts of it are exposed through a facade of APIs designed specifically for the simulation modelling of demand-responsive transport systems.

The components of the testbed can be broadly divided into three layers (see Figure 1):

- *AgentPolis Transport Simulation Model:* composed of the core simulation engine and the basic transport domain model. This model implements key elements comprising a transport system, such as road network and vehicles, and basic behavioural logic associated with them. It also provides routing algorithms and communication interfaces designed to simplify the implementation of higher-level simulation logic.
- *Testbed Core:* specializes the general AgentPolis simulator for the specific purpose of modelling demand-responsive transport systems. It implements the model of three types of agents (*Passengers*, *Drivers* and *Dispatchers*) and provides extensible abstractions for defining their behaviour.
- *Control Mechanism:* a user-supplied implementation of a specific control mechanism that is to be experimentally evaluated.

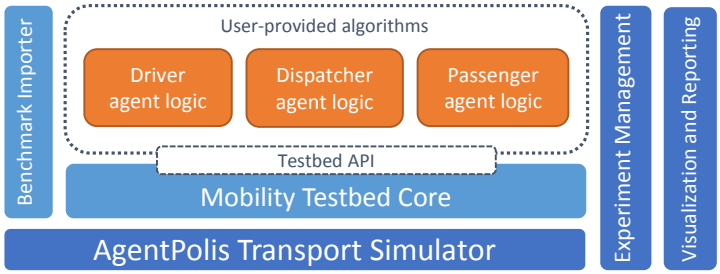

Fig. 1: Testbed's architecture overview.

In addition to these three layers, the testbed provides a suite of tools that facilitate creation, execution and evaluation of simulation experiments:

- *Benchmark importer* loads all the required input data (discussed in detail in Section 4), creates internal environment objects and agents, constructs the graph representation of a road network and simplifies it by selectively removing redundant information in order to accelerate the reasoning without losing accuracy.
- *Experiment management* is supported by a "benchmark generator" and "design of experiment" tools. The *generator* allows users to build their own benchmark scenarios (that can be imported by the importer described above) covering real-world or fictional locations with custom numbers and types of agents. Agents can be generated either based on real-world data or randomly, using various temporal and spatial distributions.
  Since a robust evaluation of the control mechanism under a sufficiently wide range of circumstances may require many simulation runs, the testbed provides tools for accelerating the evaluation process. In particular, it can use *design of experiments* methods to generate simulation configurations in a way so that maximum information about the behaviour of the control mechanism is obtained using a minimum number of simulation runs.
- The *Analysis and visualization* tools provide a way to interactively browse and review simulation execution and results at different spatial and temporal resolutions. The aggregated results, as well as the visualizations, are generated based on the detailed low-level event log recorded during the simulation, containing all the important events related to passengers (`passGotInVehicle`, `passGotOffVehicle`) and drivers with their vehicles (`vehicleMove`) and all the communication between the agents (e.g. `passSentRequest` or `requestConfirmed`).

3.3. Transport Control Mechanism

Unless the studied control mechanism has some special features, its incorporation into the testbed only requires implementing several classes and methods. For example, in the most simple case, the user only needs to extend the `DispatchingLogic` class and implement its `processNewRequest(Request r)` method.

The testbed allows us to incorporate and study a variety of control mechanisms. They can be divided into *centralized* or *decentralized* mechanisms, based on the degree of autonomy of the drivers. We also distinguish between *dynamic* and *static* control mechanisms, each suitable for the transport demand with different temporal structure.

Centralized vs. Decentralized: In a demand-responsive transport system, the behaviour of driver agents can be governed either centrally by (single or multiple) dispatcher agents, locally by the drivers themselves, or the combination of both. The reasoning logic for individual agents and central authorities is implemented by extending specific methods of abstract classes `PassengerLogic`, `DriverLogic` and `DispatchingLogic` (see Tables 1, 2 and 3). Decentralized mechanisms are suitable in situations when communication capabilities are restricted, or when the agents are independent and self-interested but can still benefit from collaboration (e.g. ridesharing[11]).

Static vs. Dynamic: Dynamic control mechanisms (sometimes called "online") process the travel demand requests when they are announced. On the other hand, static (or "offline") mechanisms need to know all the requests in advance. Our testbed grants the driver or dispatcher agents the access to requests only after they are announced by the passengers. Nevertheless, to also cater for the requirements of static mechanisms, there are several benchmarks in which the travel demand is announced long time in advance.

`sendRequest(Request r)`: Called by the testbed whenever this passenger is supposed to announce a new travel request `r`, according to input data. The passenger should contact other agents (dispatcher or drivers) within this method. `processProposal(Proposal p)`, `processRejection(RequestRejection r)`: Two methods that are called when the passenger receives a trip proposal `p` specifying details about the trip (e.g. price or arrival time) or rejection `r` of his older request from a driver or dispatcher. `vehicleArrived(String driverId, String ve-hicleId)`: Called when a driver arrives to pick the passenger up. Typically, the passenger just gets on board this driver's vehicle.

Table 1: Abstract methods of `PassengerLogic` class, implementing the behaviour of passenger agents.

`processNewRequest(Request r)`: A method called whenever the driver receives a new travel request `r` from a passenger or dispatcher. Here, the driver should react by sending his trip proposal or request rejection. `processNewAcceptance(Proposal p)`: Called when the driver's trip proposal `p` is accepted by the passenger. Here, the driver usually plans his route and starts driving. `processNewRejection(Proposal p)`: If driver's proposal `p` is rejected, the testbed calls this method. `processPassengerGotIn(String passengerId)`: Called when the passenger gets on board this driver's vehicle.

Table 2: Abstract methods of `DriverLogic` class, implementing the behaviour of driver agents.

`processNewRequest(Request r)`, `processNewAcceptance(Proposal p)`, `processNewRejection(Proposal p)`: The methods with similar meaning as in `DriverLogic` class with the exception that the dispatcher usually negotiates with passengers and only sends instructions and routes to drivers.

Table 3: Abstract methods of `DispatchingLogic` class, implementing the behaviour of dispatcher agents.

4. Experiment Process

After the tested mechanism is incorporated into the framework, the actual experimentation using the testbed follows a three-step process, as depicted in Figure Fig. 2a.

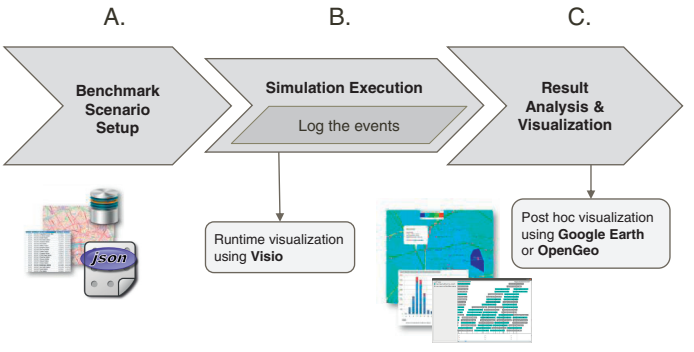

Fig. 2a : Three-step process of the experiment (setup, simulation, evaluation).

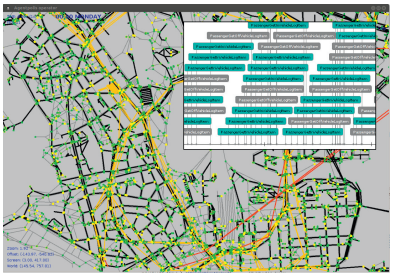

Fig. 2b : Runtime view of a running simulation. Road network, `Passenger` and `Driver` agents are shown. Simulation events are depicted in the overlay window.

A. Scenario Definition and Setup: First of all, the user needs to set up and *configure the scenario* under which he wants the control mechanism to be evaluated. The scenario is described in terms of a benchmark package, which consists of the following files:

- *Road network* – The road network in the experiment area represented in the *OpenStreetMap*[4] *(OSM)* format.
- *Driver agents* – Description (in JSON) of all the relevant drivers with their initial positions and the properties of their vehicles including the capacity, fuel consumption, CO_2 emissions or non-standard equipment (e.g. wheelchair accessibility).
- *Travel demand* – The exact representation (in JSON) of travel demand containing all the passenger agents with their associated trip details: origin and destination coordinates, time windows, announcement time and special requirements.

B. Simulation Execution: Once the model is set up, the user invokes the simulation engine to execute the simulation. The AgentPolis engine employs the discrete event simulation approach[2] in which the operation of the target system is modelled as a discrete sequence of (possibly concurrent) events in time. Each event occurs at a particular time, with precision to milliseconds of the simulation time, and marks a change of state of the modelled system. Since there are no changes occurring between consecutive events, the simulation can directly jump in time from one event to the next, which, in most cases, makes it more computationally efficient than time-stepped approaches.

The simulation progress can be presented visually during run-time, using the internal visualization component of AgentPolis. It is capable of displaying the transport network and agents within the model, along with a convenient visualization of all the ongoing events (see Figure Fig. 2b).

C. Result Analysis and Visualization: From the low-level event log recorded during the simulation run, the testbed calculates a range of higher-level, aggregate performance metrics, such as *total distance driven, fuel consumption, CO_2 emissions*, or *passenger's wait* or *travel time* statistics. Additional metrics can be defined.

In addition to low-level event logs and highly aggregated metrics, the testbed also provides the means to visualize the simulation runs and results in the geospatial and temporal context, using external tools. The interactive geobrowser *Google Earth*[5] can display the log of a simulation run exported in *Keyhole Markup Language*[6] (KML). It is capable of displaying a large number of agents, along with simple geometry and screen overlays, over a realistic satellite imagery and 3D model of the environment (see Figure Fig. 3a).

Fig. 3a : Simulation run of on-demand transport coordination scenario exported in KML format and displayed by Google Earth (based on historical data from San Francisco, 2008).

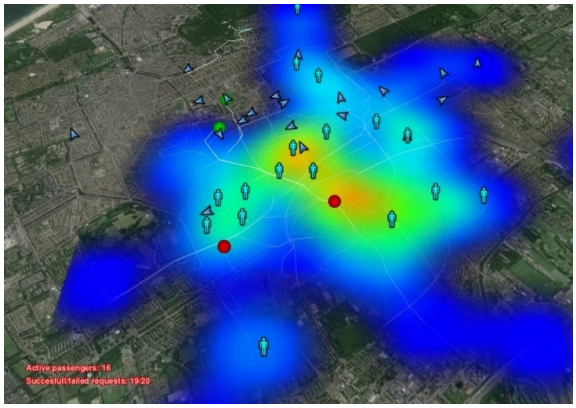

Fig. 3b : Heat map representing the spatial distribution of successfully served trip requests in Hague, Netherlands.

[4] http://openstreetmap.org/
[5] http://earth.google.com/
[6] http://developers.google.com/kml/

5. Example Study

To demonstrate how the testbed can be used, we implemented and evaluated a control mechanism for dynamic multi-vehicle dial-a-ride problem, based on *parallel tabu search heuristic*.[1] Mechanisms like this can be directly used to implement a taxi sharing service.

Using our *benchmark generator*, we prepared two collections of scenarios: one was situated in a city of Hague, Netherlands, covering the area of 81.71 km^2, while the other collection was set in Prague, Czech Republic, spread over the area of 341.03 km^2. The travel requests of passenger agents were generated with realistic temporal distribution, taking into account peak/off-peak hours, and uniform spatial distribution over the whole transport network. Each collection contained 24 scenarios – one for every combination of the number of driver agents (10 to 35 drivers, increasing by 5) and request frequency (from 100 up to 175 requests per day, increasing by 25).[7]

Since this particular control mechanism is centralized in the sense that the dispatcher agent has complete power over the behaviour of all the drivers, we only needed to extend the abstract class `DispatchingLogic` and implement its method `processNewRequest(Request r)`, which is called every time the passenger agent announces a travel request.

First, we analysed the relation between *request frequency* and *success rate*, computed as a ratio of successfully served requests and all the announced requests, with different *numbers of driver agents*.

We learned that in Prague we would need roughly 30 drivers to satisfy at least 90% of 150 daily requests, whereas in the smaller city of Hague we could maintain the same success rate with only 15 drivers (see Figure 4).

Once we had the estimation of optimal driver count, we were interested in the approximate distance driven by them on a daily basis. According to the experiments, to satisfy 150 requests per day, 30 drivers in Prague would drive 1722.29 km, while 15 drivers in Hague only 753.01 km (see Figure 5).

[7] Experiments with different control mechanism and up to 10000 requests / 500 drivers are presented in our ITSC paper from 2013[7].

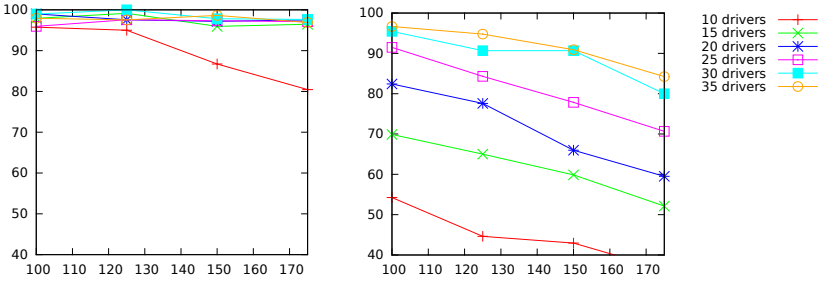

Fig. 4: Success rates in % (axis y) in relation to request frequency (axis x), with different numbers of drivers in Hague, Netherlands (left) and Prague, Czech Republic (right).

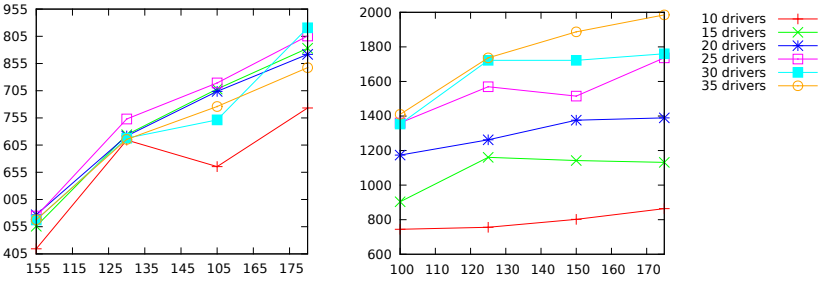

Fig. 5: Total distance in *km* driven per day (axis y) to serve increasing numbers of requests (axis x) by different numbers of agents in Hague, Netherlands (left) and Prague, Czech Republic (right).

This way, we were able to estimate daily expenses and initial investments needed to serve a specific demand in two different cities. We were also able to study a number of other metrics (see the enumeration in Subsection 4) and relations between them.

6. Conclusion

We have presented a testbed for simulation-based evaluation of on-demand mobility services. The testbed allows its users to incorporate their own control mechanisms, to evaluate them with respect to a variety of performance metrics and to compare their performance to alternative mechanisms under identical conditions using benchmark scenarios, based on realistic real-world or synthetic data. As such, the testbed can help policy makers and transport operators to assess on-demand mobility services prior to their deployment as well as it can assist researchers in developing new control mechanisms of on-demand mobility.

In the future, we aim to fully capitalize on the fact that the testbed is built on top of the versatile AgentPolis transport simulation framework. Two of the features that we plan to add in the near future are the incorporation of realistic time-dependent speed profiles for road network links and the use of activity-based models for passenger demand generation. In a longer term, we aim to combine the model of on-demand mobility services with the model of other transport modes supported by AgentPolis in order to allow studying properties of on-demand mobility within the context of fully integrated multimodal transport systems.

The testbed is freely available from `http://github.com/agents4its/mobilitytestbed/`.

7. Acknowledgements

This work was funded by Technology Agency of the Czech Republic (grant no: TE01020155), Ministry of Education, Youth and Sports of Czech Republic (grant no. 7E12065) and by the European Union Seventh Framework Programme FP7/2007-2013 (grant agreement no. 289067).

References

1. A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, 2004.
2. J. Banks, J. S. Carson, B. L. Nelson, D. M. Nicol, et al. *Discrete-event system simulation*. Pearson Prentice Hall, NJ, 2005.
3. P. M. d'Orey, R. Fernandes, and M. Ferreira. Empirical evaluation of a dynamic and distributed taxi-sharing system. In *Proceedings of ITSC 2012*, pages 140–146. IEEE, 2012.
4. L. Fu. A simulation model for evaluating advanced dial-a-ride paratransit systems. *Transportation Research Part A: Policy and Practice*, 36(4):291–307, 2002.
5. A. Glaschenko, A. Ivaschenko, G. Rzevski, and P. Skobelev. Multi-agent real time scheduling system for taxi companies. In *Proceedings of AAMAS 2009*, pages 29–36, 2009.
6. M. Horn. Multi-modal and demand-responsive passenger transport systems: a modelling framework with embedded control systems. *Transportation Research Part A*, 36(2):167–188, 2002.
7. M. Jakob and Z. Moler. Modular framework for simulation modelling of interaction-rich transport systems. In *Proceedings of ITSC 2013*. IEEE, 2013.
8. M. Jakob, Z. Moler, A. Komenda, Z. Yin, A. X. Jiang, M. P. Pěchouček, and M. Tambe. Agentpolis: towards a platform for fully agent-based modeling of multi-modal transportation. In *Proceedings of AAMAS 2012*, volume 3, pages 1501–1502, 2012.
9. M. Jakob, M. Pechoucek, M. Cap, P. Novák, and O. Vanek. Mixed-reality testbeds for incremental development of HART applications. *IEEE Intelligent Systems*, 27(2):19–25, 2012.
10. J. Jlassi, J. Euchi, and H. Chabchoub. Dial-a-ride and emergency transportation problems in ambulance services. *Computer Science and Engineering*, 2(3):17–23, 2012.
11. E. Kamar and E. Horvitz. Collaboration and shared plans in the open world: Studies of ridesharing. In *Proceedings of IJCAI 2009*, volume 9, page 187, 2009.
12. E. Lioris, G. Cohen, and A. de La Fortelle. Overview of a dynamic evaluation of collective taxi systems providing an optimal performance. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 1110–1115. IEEE, 2010.
13. M. Pěchouček, M. Jakob, and P. Novák. Towards simulation-aided design of multi-agent systems. In *Programming Multi-Agent Systems*, pages 3–21. Springer, 2012.
14. L. Quadrifoglio, M. M. Dessouky, and F. Ordóñez. A simulation study of demand responsive transit system design. *Transportation Research Part A: Policy and Practice*, 42(4):718–737, 2008.